

Try-Mopsa

Relational Static Analysis in Your Pocket

Raphaël Monat
`rmonat.fr`



Modular Open Platform for Static Analysis [Jou+19]
`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)



Modular Open Platform for Static Analysis [Jou+19]
`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

- ▶ Allows users to choose their composition of abstract domains



Modular Open Platform for Static Analysis [Jou+19]
`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

- ▶ Allows users to choose their composition of abstract domains
- ▶ Eases development of highly expressive relational static analyses



Modular Open Platform for Static Analysis [Jou+19]
`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

- ▶ Allows users to choose their composition of abstract domains
- ▶ Eases development of highly expressive relational static analyses
- ▶ Can be used as an experimentation platform



Modular Open Platform for Static Analysis [Jou+19]
`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

- ▶ Allows users to choose their composition of abstract domains
- ▶ Eases development of highly expressive relational static analyses
- ▶ Can be used as an experimentation platform
 - Supports large subsets of C, Python

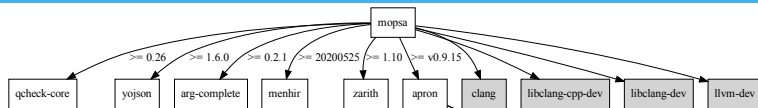


Modular Open Platform for Static Analysis [Jou+19]
`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

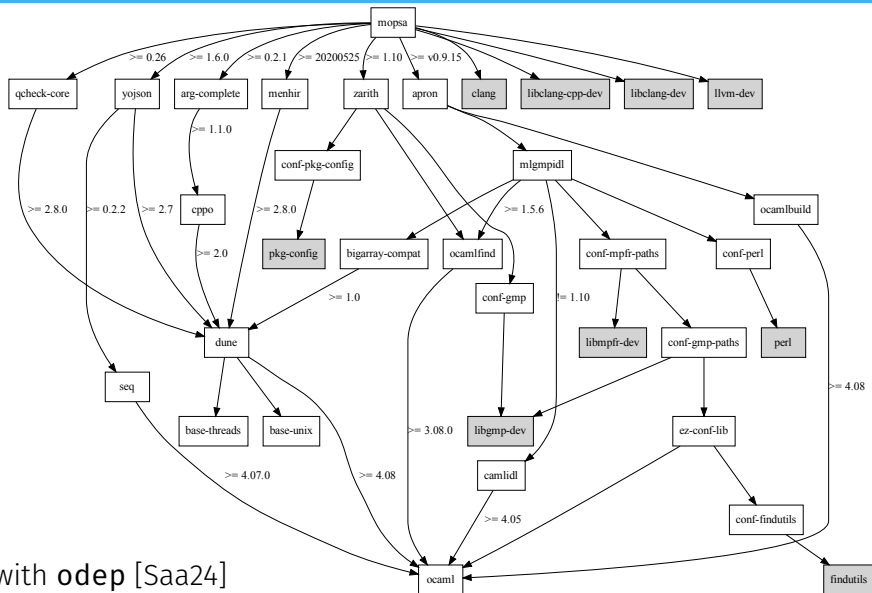
Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

- ▶ Allows users to choose their composition of abstract domains
- ▶ Eases development of highly expressive relational static analyses
- ▶ Can be used as an experimentation platform
 - Supports large subsets of C, Python
 - Academically competitive on real-world benchmarks (SV-Comp)

Static analyzers are complex pieces of software!



Static analyzers are complex pieces of software!



Try-Mopsa: a scaled down version of Mopsa running purely in browsers.

Try-Mopsa: a scaled down version of Mopsa running purely in browsers.

- ▶ Zero-install

Try-Mopsa: a scaled down version of Mopsa running purely in browsers.

- ▶ Zero-install
- ▶ Supports core features of Mopsa

Try-Mopsa: a scaled down version of Mopsa running purely in browsers.

- ▶ Zero-install
- ▶ Supports core features of Mopsa
- ▶ Features responsive interface supporting smartphones to computers

Try-Mopsa: a scaled down version of Mopsa running purely in browsers.

- ▶ Zero-install
- ▶ Supports core features of Mopsa
- ▶ Features responsive interface supporting smartphones to computers
- ▶ Scales in number of users (purely client-side)

Try-Mopsa: a scaled down version of Mopsa running purely in browsers.

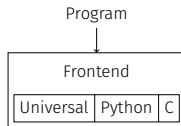
- ▶ Zero-install
- ▶ Supports core features of Mopsa
- ▶ Features responsive interface supporting smartphones to computers
- ▶ Scales in number of users (purely client-side)
- ▶ Can be convenient for onboarding or teaching?

Outline

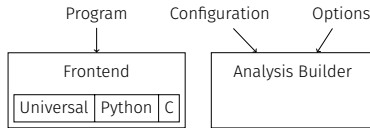
- 1 Under the Hood of Try-Mopsa
- 2 Interface Overview
- 3 Implementation Discussion
- 4 Conclusion

Under the Hood of Try-Mopsa

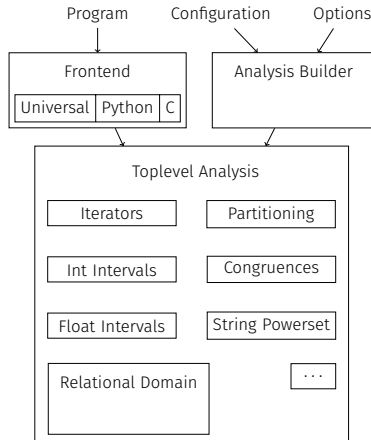
Overview of Mopsa



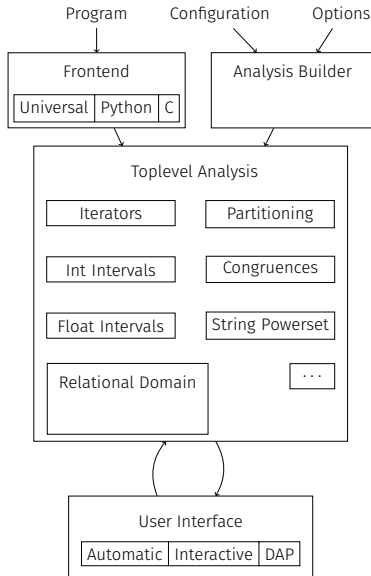
Overview of Mopsa



Overview of Mopsa



Overview of Mopsa

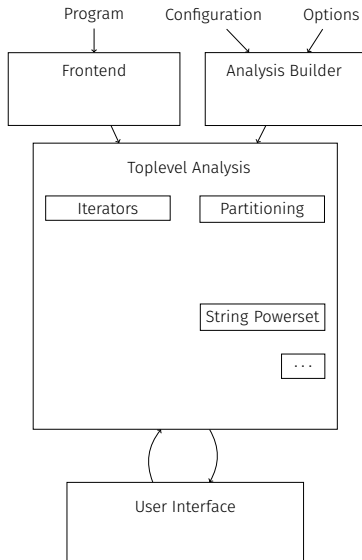


Compiling Mopsa to JavaScript

Rely on Js_of_ocaml [VB14]

89% of Mopsa's codebase is written in OCaml

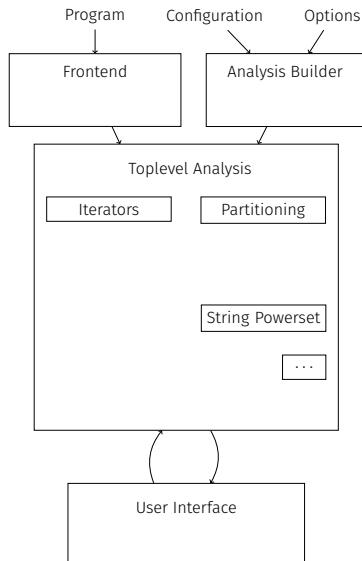
Compiling Mopsa to JavaScript



Rely on `Js_of_ocaml` [VB14]

89% of Mopsa's codebase is written in OCaml

Compiling Mopsa to JavaScript



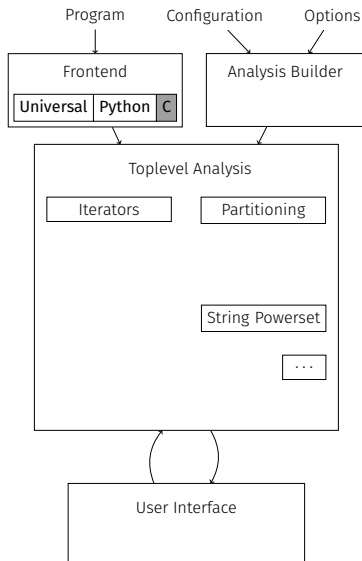
Rely on Js_of_ocaml [VB14]

Parsing libraries

Menhir

libclang: too much manual work for now

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

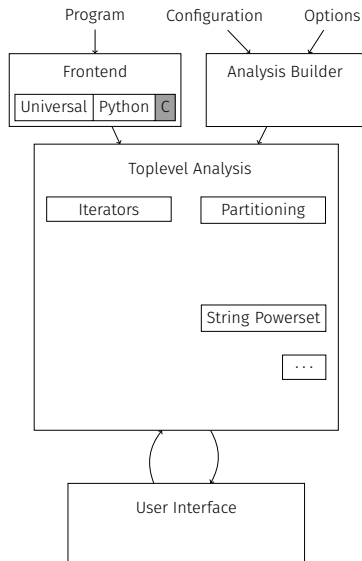
Rely on `Js_of_ocaml` [VB14]

Parsing libraries

Menhir

~~libclang~~: too much manual work for now

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

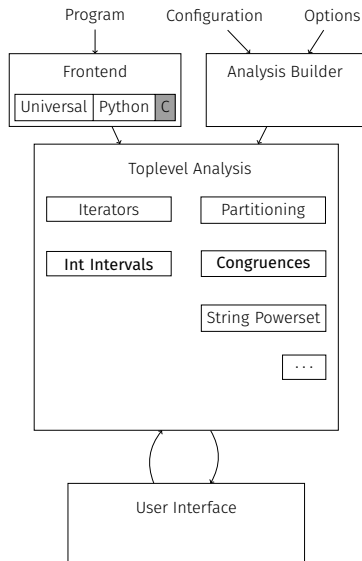
Rely on `Js_of_ocaml` [VB14]

Parsing libraries

Arbitrary-precision integer arithmetic

~~Zarith~~ $\rightsquigarrow$ `Zarith_stubs_js`

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

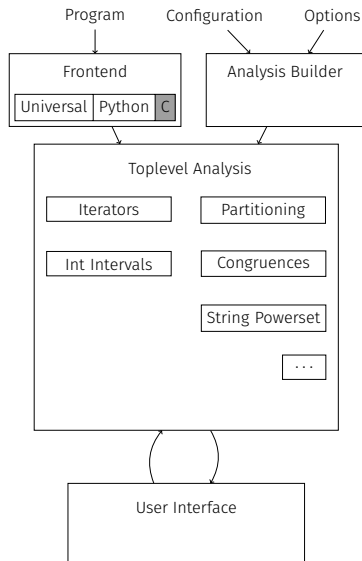
Rely on Js_of_ocaml [VB14]

Parsing libraries

Arbitrary-precision integer arithmetic

~~Zarith~~ $\rightsquigarrow$ Zarith_stubs_js

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

Rely on Js_of_ocaml [VB14]

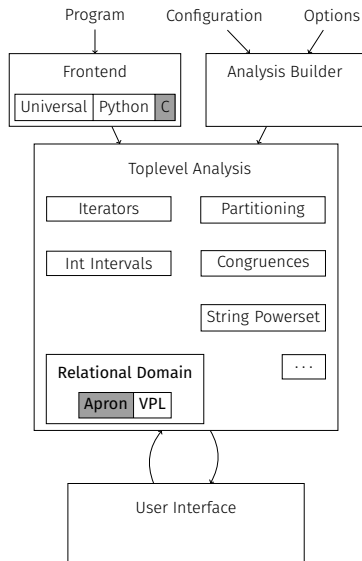
Parsing libraries

Arbitrary-precision integer arithmetic

Relational domains

Apron [JM09] $\rightsquigarrow$ VPL [Bou+18] w/ extensions

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

Rely on Js_of_ocaml [VB14]

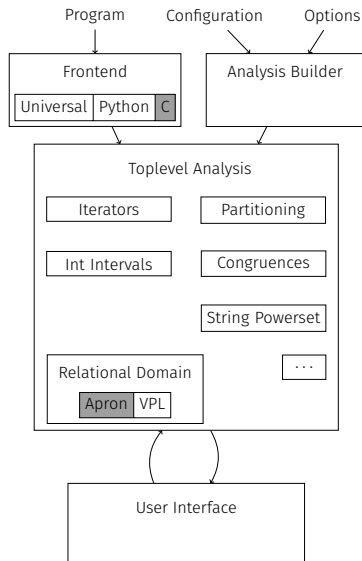
Parsing libraries

Arbitrary-precision integer arithmetic

Relational domains

Apron [JM09] $\rightsquigarrow$ VPL [Bou+18] w/ extensions

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

Rely on Js_of_ocaml [VB14]

Parsing libraries

Arbitrary-precision integer arithmetic

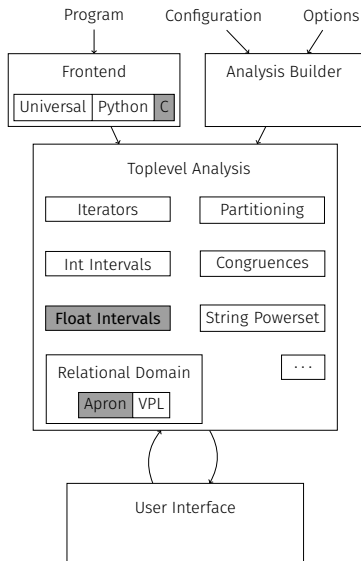
Relational domains

~~Floating-point~~

Our implementation needs to set rounding modes

Not supported in JavaScript/WebAssembly.

Compiling Mopsa to JavaScript



Gray = unsupported by Try-Mopsa.

Rely on Js_of_ocaml [VB14]

Parsing libraries

Arbitrary-precision integer arithmetic

Relational domains

~~Floating-point~~

Our implementation needs to set rounding modes

Not supported in JavaScript/WebAssembly.

Toplevel

- ▶ Handles user actions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS
- ▶ Intercepts `std{in,out}`, forwards them to the toplevel

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS
- ▶ Intercepts `std{in,out}`, forwards them to the toplevel
- ▶ Relies on virtual filesystem to embed stubs and configurations

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS
- ▶ Intercepts `std{in,out}`, forwards them to the toplevel
- ▶ Relies on virtual filesystem to embed stubs and configurations

Handling User Inputs

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS
- ▶ Intercepts `std{in,out}`, forwards them to the toplevel
- ▶ Relies on virtual filesystem to embed stubs and configurations

Handling User Inputs

- ▶ Mopsa CLI: synchronous, blocking interaction loop

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS
- ▶ Intercepts `std{in,out}`, forwards them to the toplevel
- ▶ Relies on virtual filesystem to embed stubs and configurations

Handling User Inputs

- ▶ Mopsa CLI: synchronous, blocking interaction loop
- ▶ But web interfaces require asynchronous interfaces!

Handling Web Interactions

Toplevel

- ▶ Handles user actions
- ▶ Ace editor with custom highlighting and ANSI rendering
- ▶ Dynamic option generation
- ▶ Triggers/interrupts web worker

Web Worker

- ▶ Contains Mopsa in JS
- ▶ Intercepts `std{in,out}`, forwards them to the toplevel
- ▶ Relies on virtual filesystem to embed stubs and configurations

Handling User Inputs

- ▶ Mopsa CLI: synchronous, blocking interaction loop
 - ▶ But web interfaces require asynchronous interfaces!
- ⇒ Rely on **sync-message** [Moj23] (low-level synchronous comm.)

Interface Overview

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8
9 assert(0 <= j < |s|  $\implies$  s[j] - 'a' < |s|);
```

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8
9 assert(0 <= j < |s|  $\implies$  s[j] - 'a' < |s|);
```

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8
9 assert(0 <= j < |s|  $\implies$  s[j] - 'a' < |s|);
```

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8
9 assert(0 <= j < |s|  $\implies$  s[j] - 'a' < |s|);
```

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8
9 assert(0 <= j < |s|  $\implies$  s[j] - 'a' < |s|);
```

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8 // s ∈ {"a", "ab", "abc", ..., "abc...xy", "abc...xyz"}
9 assert(0 <= j < |s| ⇒ s[j] - 'a' < |s|);
```

Running example

```
1 str s = "a";
2 int i = 1;
3 while('a' + i <= 'z') {
4     if (rand(0, 1)) break;
5     s = s @ to_string('a' + i);
6     i = i + 1;
7 }
8 // s ∈ {"a", "ab", "abc", ..., "abc...xy", "abc...xyz"}
9 assert(0 <= j < |s| ⇒ s[j] - 'a' < |s|);
```

⇒ try-mopsa.rmonat.fr

Implementation Discussion

- ▶ Mopsa $\simeq 87k$ LOC

Implementation

- ▶ Mopsa $\simeq 87k$ LOC
- ▶ Try-Mopsa

Implementation

- ▶ Mopsa $\simeq 87k$ LOC
- ▶ Try-Mopsa
 - 1,500 lines of HTML/CSS/JS

- ▶ Mopsa $\simeq$ 87kLOC
- ▶ Try-Mopsa
 - 1,500 lines of HTML/CSS/JS
 - 670 lines of OCaml (toplevel/worker)

- ▶ Mopsa $\simeq$ 87kLOC
- ▶ Try-Mopsa
 - 1,500 lines of HTML/CSS/JS
 - 670 lines of OCaml (toplevel/worker)
 - 500 lines of OCaml (VPL)

- ▶ Mopsa $\simeq$ 87kLOC
- ▶ Try-Mopsa
 - 1,500 lines of HTML/CSS/JS
 - 670 lines of OCaml (toplevel/worker)
 - 500 lines of OCaml (VPL)
 - Compiled, optimized size: 3.1 megabytes

- ▶ Mopsa $\simeq$ 87kLOC
- ▶ Try-Mopsa
 - 1,500 lines of HTML/CSS/JS
 - 670 lines of OCaml (toplevel/worker)
 - 500 lines of OCaml (VPL)
 - Compiled, optimized size: 3.1 megabytes
 - *No breaking changes*

Performance

Program	Binary execution	Firefox execution
attributes.py	0.03s ± 0.01	0.13s ± 0.01
fspath.py	0.03s ± 0.01	0.12s ± 0.01
list.py	0.03s ± 0.01	0.13s ± 0.01
loop.py	0.03s ± 0.01	0.13s ± 0.01
recenc.py	0.03s ± 0.01	0.13s ± 0.01
str_alphabet.u	0.04s ± 0.01	0.20s ± 0.01
str_alphabet2.u	0.23s ± 0.01	0.81s ± 0.01
str_conc_loop.u	0.08s ± 0.01	0.29s ± 0.01
str_conc_loop2.u	0.04s ± 0.01	0.15s ± 0.01

- ▶ 5× slowdown compared to native binary
- ▶ Raw performance is not a priority
- ▶ Still reasonable analysis times

Playwright framework to assess the browser compatibility of Try-Mopsa

Conformance tests

5 tests x (3 desktop browsers + 2 mobile browsers x 2 viewports)

Identified several rendering issues on mobile

Playwright framework to assess the browser compatibility of Try-Mopsa

Conformance tests

5 tests x (3 desktop browsers + 2 mobile browsers x 2 viewports)

Identified several rendering issues on mobile

Additional tests

Universal: 13 programs x 11 configurations

Python: 5 programs x 6 configurations

Browser compatibility (II)

The screenshot displays the Playwright Test interface on the left and the Mopsa analysis results on the right.

Playwright Test Results:

- Filter (e.g. text, @tag)
- Status: all Projects: chromium firefox webkit Mobile Chrome Mobile C...
- 7/7 passed (100%)
- Test results for various JSON files and actions:

 - str_alpha2u string_product_relational.json: Passed, 9.3s
 - str_conc_loop.u intervals.json: Passed, 3.1s
 - str_conc_loop.u string_product_relational.json: Passed, 1.8s
 - str_conc_loop2.u string_product_relational.json: Passed, 1.5s
 - str_dot_at_end.u string_product_relational.json: Passed, 121ms
 - str_double.u intervals.json: Passed, 182ms
 - str_double.u string_product_relational.json: Passed, 822ms
 - str_double2.u intervals.json: Passed, 48ms
 - str_double2.u string_product_relational.json: Passed, 40ms
 - attributes.py values-relational.json: Passed, 38ms
 - attributes.py values.json: Passed, 38ms
 - fspath.py values-relational.json: Passed, 33ms
 - fspath.py values.json: Passed, 40ms
 - list.py values-relational.json: Passed, 33ms
 - list.py values.json: Passed, 41ms
 - loop.py values-relational.json: Passed, 39ms
 - loop.py values.json: Passed, 37ms
 - recency.py values-relational.json: Passed, 36ms
 - recency.py values.json: Passed, 49ms
 - conformance.spec.ts: Passed, 47ms
 - universal_default_tlv: Passed, 50ms
 - universal_default_rel: Passed, 70ms
 - universal_default_rel_interactive: Passed, 49.8s
 - chromium: Passed, 6.0s
 - Firefox: Passed, 8.0s
 - webkit: Passed, 6.1s
 - Mobile Chrome: Passed, 5.7s
 - Mobile Chrome Landscape: Passed, 8.2s
 - Mobile Safari: Passed, 8.2s
 - Mobile Safari Landscape: Passed, 8.2s
 - error_message: Passed
 - share_button: Passed

Mopsa Analysis:

- Code Editor: Universal Python
- Code Editor Content:

```
1 int i = 0;
2 int j = rand(10, 20);
3 while (i < j) {
4     i = i + 1;
5 }
6 print();
7 assert(i == j);
```

- Mopsa Analysis Output:

```
25 7 assert(i == j);
26 8
27 1 i = (i + 1);
28 input.u:4.4-14
29 1 int i = 0;
30 2 int j = rand(10, 20);
31 3 while (i < j) {
32 4     i = i + 1;
33 5 }
34 6 print();
35 7 assert(i == j);
36 8
37 1 i = (i + 1);
38 input.u:4.4-14
39 1 int i = 0;
40 2 int j = rand(10, 20);
41 3 while (i < j) {
42 4     i = i + 1;
43 5 }
44 6 print();
45 7 assert(i == j);
46 8
47 1 i = (i + 1);
48 int-iti U strings :
49 i : [1,19] ,
50 numeric-relations : [ 1 ≤ i, 10 ≤ j, j ≤ 20, i + 1 ≤ j ]
51 mopsa ==
```

No errors

Conclusion

- ▶ Server-side: Interproc [Jea09], Banal [Min12], FuncTion [Urb15]
Less adaptation required, but maintenance and security issues

- ▶ Server-side: Interproc [Jea09], Banal [Min12], FuncTion [Urb15]
Less adaptation required, but maintenance and security issues
- ▶ Client-side: Ciao Prolog [MPH22], Salto (OCaml) [LM25] Binsec tutorial [RB25]
Try-Mopsa supports relational domains and interactive user input

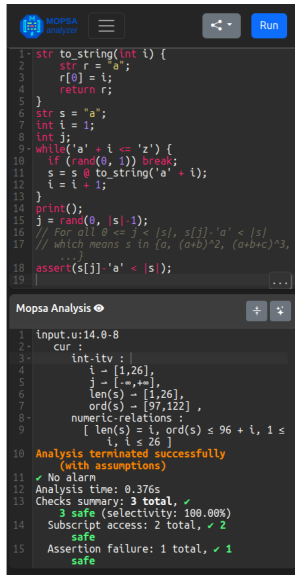
- ▶ Server-side: Interproc [Jea09], Banal [Min12], FuncTion [Urb15]
Less adaptation required, but maintenance and security issues
- ▶ Client-side: Ciao Prolog [MPH22], Salto (OCaml) [LM25] Binsec tutorial [RB25]
Try-Mopsa supports relational domains and interactive user input
- ▶ LiSA for teaching [Neg+24]

- ▶ Server-side: Interproc [Jea09], Banal [Min12], FuncTion [Urb15]
Less adaptation required, but maintenance and security issues
- ▶ Client-side: Ciao Prolog [MPH22], Salto (OCaml) [LM25] Binsec tutorial [RB25]
Try-Mopsa supports relational domains and interactive user input
- ▶ LiSA for teaching [Neg+24]
- ▶ Interaction with IDEs [LDB19]

- ▶ Server-side: Interproc [Jea09], Banal [Min12], FuncTion [Urb15]
Less adaptation required, but maintenance and security issues
- ▶ Client-side: Ciao Prolog [MPH22], Salto (OCaml) [LM25] Binsec tutorial [RB25]
Try-Mopsa supports relational domains and interactive user input
- ▶ LiSA for teaching [Neg+24]
- ▶ Interaction with IDEs [LDB19]
- ▶ Works between concrete and abstract debuggers [Do+20; Hol+24; MVR23]

Conclusion

► Pure JS version of Mopsa



The screenshot displays the Mopsa analyzer interface. At the top, there's a header with the Mopsa logo, a menu icon, a share icon, and a 'Run' button. Below the header is a code editor with a dark theme. The code defines a function `to_string` that takes an integer `i` and returns a string `r` of length `i` filled with 'a's. The main code block initializes `s` to "a", sets `i` to 1, and enters a `while` loop that continues as long as `'a' + i` is less than or equal to 'z'. Inside the loop, it randomly breaks or appends a character to `s` and increments `i`. After the loop, it prints `s` and asserts that the last character of `s` is 'a'.

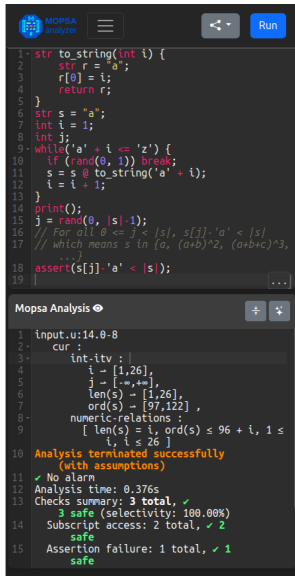
Below the code editor is a section titled 'Mopsa Analysis' with expand/collapse and zoom icons. It shows the analysis results for the provided code. The input is `u:14.0-8`. The analysis includes details about the current state (e.g., `int-itv`, `len(s)`, `ord(s)`) and concludes with a successful termination message, indicating no alarm was raised and providing a summary of checks performed.

```
1- str to_string(int i) {
2-   str r = "a";
3-   r[0] = i;
4-   return r;
5- }
6- str s = "a";
7- int i = 1;
8- int j;
9- while('a' + i <= 'z') {
10-   if (rand(0, 1)) break;
11-   s = s @ to_string('a' + i);
12-   i = i + 1;
13- }
14- print();
15- j = rand(0, |s|-1);
16- // For all 0 <= j < |s|, s[j]·'a' < |s|
17- // which means s in {a, (a+b)^2, (a+b+c)^3, ...}
18- assert(s[j]·'a' < |s|);
19- ...
```

Mopsa Analysis

```
1 input.u:14.0-8
2 cur :
3 int-itv : |
4 i - [1,26],
5 j - [·,·,·],
6 len(s) - [1,26],
7 ord(s) - [97,122] ,
8 numeric-relations :
9 [ len(s) = i, ord(s) ≤ 96 + i, 1 ≤ i, i ≤ 26 ]
10 Analysis terminated successfully
   (with assumptions)
11 ✓ No alarm
12 Analysis time: 0.376s
13 Checks summary: 3 total, ✓
   3 safe (selectivity: 100.00%)
   Subscript access: 2 total, ✓ 2
   safe
14 Assertion failure: 1 total, ✓ 1
   safe
```

Conclusion



The screenshot displays the Mopsa analyzer interface. At the top, there's a header with the Mopsa logo, a menu icon, a share icon, and a 'Run' button. Below the header is a code editor with a JavaScript snippet. The code defines a function `to_string` that takes an integer `i` and returns a string `r` of length `i` filled with 'a's. It then initializes `s` to "a", sets `i` to 1, and enters a `while` loop that continues as long as `'a' + i` is less than or equal to 'z'. Inside the loop, it randomly breaks or appends a character to `s` and increments `i`. After the loop, it prints `s` and asserts that the last character of `s` is 'a'.

Below the code editor is a panel titled 'Mopsa Analysis' which shows the analysis results. It includes input values, a summary of the analysis, and a list of checks.

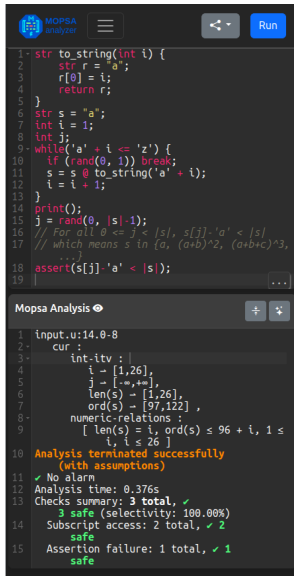
```
1- str to_string(int i) {
2-   str r = "a";
3-   r[0] = i;
4-   return r;
5- }
6- str s = "a";
7- int i = 1;
8- int j;
9- while('a' + i <= 'z') {
10-   if (rand(0, 1)) break;
11-   s = s @ to_string('a' + i);
12-   i = i + 1;
13- }
14- print();
15- j = rand(0, |s|-1);
16- // For all 0 <= j < |s|, s[j]·'a' < |s|
17- // which means s in {a, (a+b)^2, (a+b+c)^3, ...}
18- assert(s[j]·'a' < |s|);
19- }
```

Mopsa Analysis

```
1 input.u:14.0-8
2 cur :
3 int-ity : |
4 i - [1,26],
5 j - [·,·,·],
6 len(s) - [1,26],
7 ord(s) - [97,122] ,
8 numeric-relations :
9 [ len(s) = i, ord(s) ≤ 96 + i, 1 ≤ i, i ≤ 26 ]
10 Analysis terminated successfully
    (with assumptions)
11 ✓ No alarm
12 Analysis time: 0.376s
13 Checks summary: 3 total, ✓
    3 safe (selectivity: 100.00%)
14 Subscript access: 2 total, ✓ 2
    safe
15 Assertion failure: 1 total, ✓ 1
    safe
```

- ▶ Pure JS version of Mopsa
- ▶ Supports polyhedra, abstract debugger

Conclusion



The screenshot displays the Mopsa analyzer interface. At the top, there's a header with the Mopsa logo, a menu icon, a share icon, and a 'Run' button. Below the header, a code editor shows a JavaScript function `to_string` and a `while` loop that builds a string `s` from random characters. The code includes an assertion `assert(s[j] - 'a' < |s|);`. Below the code editor, the 'Mopsa Analysis' panel shows the analysis results. It starts with 'input.u:14.0-8' and 'cur :'. The analysis includes variable ranges for `i`, `j`, `len(s)`, and `ord(s)`, and numeric relations for `len(s)` and `i`. The final summary states: 'Analysis terminated successfully (with assumptions)', 'No alarm', 'Analysis time: 0.376s', 'Checks summary: 3 total, 3 safe (selectivity: 100.00%)', 'Subscript access: 2 total, 2 safe', and 'Assertion failure: 1 total, 1 safe'.

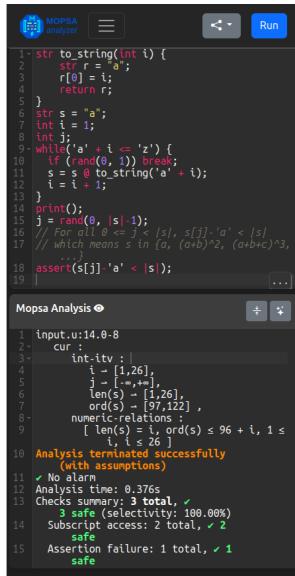
```
1- str to_string(int i) {
2-   str r = "a";
3-   r[0] = i;
4-   return r;
5- }
6- str s = "a";
7- int i = 1;
8- int j;
9- while('a' + i <= 'z') {
10-   if (rand(0, 1)) break;
11-   s = s @ to_string('a' + i);
12-   i = i + 1;
13- }
14- print();
15- j = rand(0, |s|-1);
16- // For all 0 <= j < |s|, s[j] - 'a' < |s|
17- // which means s in {a, (a+b)^2, (a+b+c)^3, ...}
18- assert(s[j] - 'a' < |s|);
19- }
```

Mopsa Analysis

```
1 input.u:14.0-8
2 cur :
3 int-ityv : |
4   i - [1,26],
5   j - [0,26],
6   len(s) - [1,26],
7   ord(s) - [97,122] ,
8 numeric-relations :
9   [ len(s) = i, ord(s) ≤ 96 + i, 1 ≤ i, i ≤ 26 ]
10 Analysis terminated successfully
11    (with assumptions)
12 ✓ No alarm
13 Analysis time: 0.376s
14 Checks summary: 3 total, ✓ 3
15    3 safe (selectivity: 100.00%)
16 Subscript access: 2 total, ✓ 2
17    safe
18 Assertion failure: 1 total, ✓ 1
19    safe
```

- ▶ Pure JS version of Mopsa
- ▶ Supports polyhedra, abstract debugger
- ▶ Works on smartphones and desktops alike

Conclusion



The screenshot displays the Mopsa analyzer interface. At the top, there's a header with the Mopsa logo, a menu icon, a share icon, and a 'Run' button. Below the header is a code editor with the following JavaScript code:

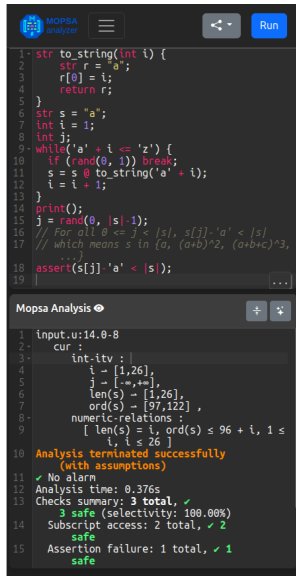
```
1- str to_string(int i) {  
2-   str r = "a";  
3-   r[0] = i;  
4-   return r;  
5- }  
6- str s = "a";  
7- int i = 1;  
8- int j;  
9- while('a' + i <= 'z') {  
10-   if (rand(0, 1)) break;  
11-   s = s @ to_string('a' + i);  
12-   i = i + 1;  
13- }  
14- print();  
15- j = rand(0, |s|-1);  
16- // For all 0 <= j < |s|, s[j]·'a' < |s|  
17- // which means s in {a, (a+b)^2, (a+b+c)^3,  
18-   ...}  
19- assert(s[j]·'a' < |s|);
```

Below the code editor is a panel titled 'Mopsa Analysis' with a zoom-in and zoom-out icon. It contains the following analysis results:

```
1 input.u:14.0-8  
2 cur :  
3 int-ity : |  
4 i - [1,26],  
5 j - [·,·,·],  
6 len(s) - [1,26],  
7 ord(s) - [97,122] ,  
8 numeric-relations :  
9 [ len(s) = i, ord(s) ≤ 96 + i, 1 ≤  
   i, i ≤ 26 ]  
10 Analysis terminated successfully  
   (with assumptions)  
11 ✓ No alarm  
12 Analysis time: 0.376s  
13 Checks summary: 3 total, ✓  
   3 safe (selectivity: 100.00%)  
14 Subscript access: 2 total, ✓ 2  
   safe  
15 Assertion failure: 1 total, ✓ 1  
   safe
```

- ▶ Pure JS version of Mopsa
- ▶ Supports polyhedra, abstract debugger
- ▶ Works on smartphones and desktops alike
- ▶ Useful for demo or lightweight teaching purposes

Conclusion



The screenshot displays the Mopsa analyzer interface. At the top, there's a header with the Mopsa logo, a menu icon, a share icon, and a 'Run' button. Below the header, a code editor shows a JavaScript function `to_string` and a `while` loop. The code is as follows:

```
1- str to_string(int i) {  
2-   str r = "a";  
3-   r[0] = i;  
4-   return r;  
5- }  
6- str s = "a";  
7- int i = 1;  
8- int j;  
9- while('a' + i <= 'z') {  
10-   if (rand(0, 1)) break;  
11-   s = s @ to_string('a' + i);  
12-   i = i + 1;  
13- }  
14- print();  
15- j = rand(0, |s|-1);  
16- // For all 0 <= j < |s|, s[j]·'a' < |s|  
17- // which means s in {a, (a+b)^2, (a+b+c)^3,  
18- // ...}  
19- assert(s[j]·'a' < |s|);  
20- }
```

Below the code editor, the 'Mopsa Analysis' panel shows the results of the analysis. It includes input values, a summary of the analysis, and a list of checks.

```
1 input.u:14.0-8  
2 cur :  
3 int-ity : |  
4 i - [1,26],  
5 j - [·,·,·],  
6 len(s) - [1,26],  
7 ord(s) - [97,122],  
8 numeric-relations :  
9 [ len(s) = i, ord(s) ≤ 96 + i, 1 ≤  
10 i, i ≤ 26 ]  
11 Analysis terminated successfully  
12 (with assumptions)  
13 ✓ No alarm  
14 Analysis time: 0.376s  
15 Checks summary: 3 total, ✓  
16 3 safe (selectivity: 100.00%)  
17 Subscript access: 2 total, ✓ 2  
18 safe  
19 Assertion failure: 1 total, ✓ 1  
20 safe
```

- ▶ Pure JS version of Mopsa
- ▶ Supports polyhedra, abstract debugger
- ▶ Works on smartphones and desktops alike
- ▶ Useful for demo or lightweight teaching purposes

`try-mopsa.rmonat.fr`

References – I

- [Bou+18] Sylvain Boulmé et al. **“The Verified Polyhedron Library: an Overview”**. In: IEEE, 2018, pp. 9–17. DOI: [10.1109/SYNASC.2018.00014](https://doi.org/10.1109/SYNASC.2018.00014).
- [Do+20] Lisa Nguyen Quang Do et al. **“Debugging Static Analysis”**. In: IEEE Trans. Software Eng. 7 (2020), pp. 697–709. DOI: [10.1109/TSE.2018.2868349](https://doi.org/10.1109/TSE.2018.2868349).
- [Hol+24] Karoliine Holter et al. **“Abstract Debuggers: Exploring Program Behaviors using Static Analysis Results”**. In: ed. by Jonathan Edwards and Marcel Taeumel. ACM, 2024, pp. 130–146. DOI: [10.1145/3689492.3690053](https://doi.org/10.1145/3689492.3690053).

References – II

- [Jea09] Bertrand Jeannet. Web Interface for the Interproc Analyzer. 2009. URL: <https://pop-art.inrialpes.fr/interproc/interprocweb.cgi.html>.
- [JM09] Bertrand Jeannet and Antoine Miné. **“Apron: A Library of Numerical Abstract Domains for Static Analysis”**. In: Springer, 2009, pp. 661–667.
- [Jou+19] M. Journault et al. **“Combinations of reusable abstract domains for a multilingual static analyzer”**. In: New York, USA, July 2019, pp. 1–17.

References – III

- [LDB19] Linghui Luo, Julian Dolby, and Eric Bodden. **“MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper)”**. In: ed. by Alastair F. Donaldson. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 21:1–21:25. DOI: `10.4230/LIPICS.ECOOP.2019.21`.
- [LM25] Pierre Lermusiaux and Benoît Montagu. **Web Demo for the Salto Analyzer**. 2025. URL: `https://salto.gitlabpages.inria.fr/demo/salto.html`.
- [Min12] Antoine Miné. **Web Interface for Sufficient Condition Polyhedral Prototype Analyzer**. 2012. URL: `https://mine.perso.lip6.fr/banal/`.

References – IV

- [Moj23] Alex Mojaki. **sync-message**. Version 0.0.12. Oct. 2023.
- [MPH22] Jose F. Morales, Guillermo García Pradales, and Manuel Hermenegildo. **Ciao Prolog Playgrund**. 2022. URL: <https://ciao-lang.org/playground/>.
- [MVR23] Mats Van Molle, Bram Vandenbogaerde, and Coen De Roover. **“Cross-Level Debugging for Static Analysers”**. In: ed. by João Saraiva, Thomas Degueule, and Elizabeth Scott. ACM, 2023, pp. 138–148. DOI: [10.1145/3623476.3623512](https://doi.org/10.1145/3623476.3623512).

References – V

- [Neg+24] Luca Negrini et al. **“Teaching Through Practice: Advanced Static Analysis with LiSA”**. In: ed. by Emil Sekerinski and Leila Ribeiro. Lecture Notes in Computer Science. Springer, 2024, pp. 43–57. DOI: [10.1007/978-3-031-71379-8_3](https://doi.org/10.1007/978-3-031-71379-8_3).
- [RB25] Frédéric Recoules and Sébastien Bardin.
BINSEC Tutorial at PLDI’25: Adapting Symbolic Execution for Binary-level Security
2025. URL: <https://binsec.github.io/tutorial-pldi2025/>.
- [Saa24] Simmo Saan.
Odep: Dependency graphs for OCaml modules, libraries and packages.
Version 0.2.1. Apr. 2024.

References – VI

- [Urb15] Caterina Urban. **Web Interface for FuncTion**. 2015. URL: <https://www.di.ens.fr/~urban/FuncTion.html>.
- [VB14] Jérôme Vouillon and Vincent Balat. **“From bytecode to JavaScript: the Js_of_ocaml compiler”**. In: Softw. Pract. Exp. 8 (2014), pp. 951–972. DOI: 10.1002/SPE.2187.